




REPORT DOCUMENTATION PAGE

Form Approved
OPM No.

2

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of Management and Budget, Washington, DC 20503.

1. AGENCY USE (Leave)		2. REPORT		3. REPORT TYPE AND DATES	
4. TITLE AND NOS/VE Ada, Version 1.4, Host: CYBER 180-930-31 under NOS/VE Level 826, Target: SAME AS HOST 931217S1.11336				5. FUNDING DTIC SELECTE FEB 22 1994 C D	
6. Authors: National Institute of Standards and Technology Gaithersburg, Maryland					
7. PERFORMING ORGANIZATION NAME(S) AND National Institute of Standards and Technology Building 255, Room A266 Gaithersburg, Maryland 20899 USA				8. PERFORMING ORGANIZATION	
9. SPONSORING/MONITORING AGENCY NAME(S) AND Ada Joint Program Office The Pentagon, Rm 3E118 Washington, DC 20301-3080				10. SPONSORING/MONITORING AGENCY AD-A275 977 	
11. SUPPLEMENTARY					
12a. DISTRIBUTION/AVAILABILITY Approved for Public Release; distribution unlimited				12b. DISTRIBUTION	
13. (Maximum 200 NOS/VE Ada, Version 1.4, Host: CYBER 180-930-31 under NOS/VE, Level 826 Target: SAME AS HOST 931217S1.11336  94-05551  DTIC QUALITY ASSURED					
14. SUBJECT Ada programming language, Ada Compiler Validation Summary Report, Ada Compiler Val. Capability Val. Testing, Ada Val. Office, Ada Val. Facility ANSI/MIL-STD-1815A, AJPO				15. NUMBER OF	
17. SECURITY CLASSIFICATION UNCLASSIFIED		18. SECURITY UNCLASSIFIED		16. PRICE UNCLASSIFIED	
19. SECURITY CLASSIFICATION UNCLASSIFIED		20. LIMITATION OF UNCLASSIFIED			

NSN

0 4 2 18 183

Standard Form 298, (Rev. 2-89)
Prescribed by ANSI Std.

**Best
Available
Copy**

AVF Control Number: NIST93CDS500_1_1.11

DATE COMPLETED

BEFORE ON-SITE: 93-12-10

AFTER ON-SITE: 93-12-20

REVISIONS: 94-01-14

Ada COMPILER

VALIDATION SUMMARY REPORT:

Certificate Number: 931217S1.11336

Control Data Systems, Inc.

NOS/VE Ada, Version 1.4

CYBER 180-930-31 => CYBER 180-930-31

Prepared By:

Software Standards Validation Group
Computer Systems Laboratory
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899
U.S.A.

Accession For	
NTIS CRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input checked="" type="checkbox"/>
Unannounced	<input checked="" type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

AVF Control Number: NIST93CDS500_1_1.11

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on December 17, 1993.

Compiler Name and Version: NOS/VE Ada, Version 1.4

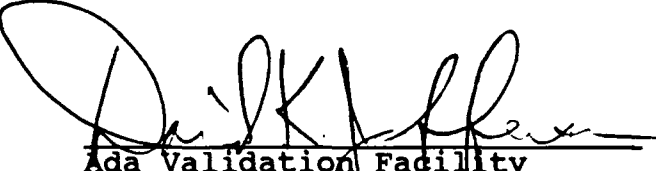
Host Computer System: CYBER 180-930-31 under NOS/VE, Level 826

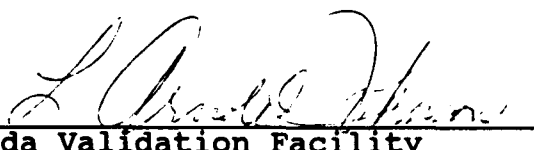
Target Computer System: CYBER 180-930-31 under NOS/VE, Level 826

See section 3.1 for any additional information about the testing environment.

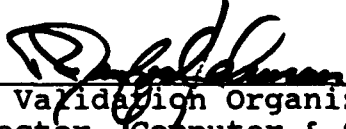
As a result of this validation effort, Validation Certificate 931217S1.11336 is awarded to Control Data Systems, Inc.. This certificate expires 2 years after ANSI/MIL-STD-1815B is approved by ANSI.

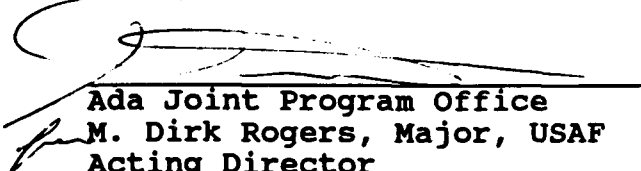
This report has been reviewed and is approved.


Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division (ISED)


Ada Validation Facility
Mr. L. Arnold Johnson
Manager, Software Standards
Validation Group

Computer Systems Laboratory (CSL)
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899
U.S.A.


for Ada Validation Organization
Director, Computer & Software
Engineering Division
Institute for Defense Analyses
Alexandria VA 22311
U.S.A.


Ada Joint Program Office
M. Dirk Rogers, Major, USAF
Acting Director
Ada Joint Program Office
Washington DC 20301
U.S.A.

NIST93CDS500_1_1.11

DECLARATION OF CONFORMANCE

The following declaration of conformance was supplied by the customer.

Customer: Control Data Systems, Inc.

Certificate Awardee: Control Data Systems, Inc.

Ada Validation Facility: National Institute of Standards and
Technology
Computer Systems Laboratory (CSL)
Software Standards Validation Group
Building 225, Room A266
Gaithersburg, Maryland 20899
U.S.A.

ACVC Version: 1.11

Ada Implementation:

Compiler Name and Version: NOS/VE Ada, Version 1.4

Host Computer System: CYBER 180-930-31 under NOS/VE,
Level 826

Target Computer System: CYBER 180-930-31 under NOS/VE,
Level 826

Declaration:

I the undersigned, declare that I have no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A ISO 8652-1987 in the implementation listed above.

James L. Hines
Customer Signature
Company Control Data Systems, Inc.
Title

12/16/93
Date

James L. Hines
Certificate Awardee Signature
Company Control Data Systems, Inc.
Title

12/16/93
Date

TABLE OF CONTENTS

CHAPTER 1.....	1-1
INTRODUCTION.....	1-1
1.1 USE OF THIS VALIDATION SUMMARY REPORT.....	1-1
1.2 REFERENCES.....	1-2
1.3 ACVC TEST CLASSES.....	1-2
1.4 DEFINITION OF TERMS.....	1-3
CHAPTER 2.....	2-1
IMPLEMENTATION DEPENDENCIES.....	2-1
2.1 WITHDRAWN TESTS.....	2-1
2.2 INAPPLICABLE TESTS.....	2-1
2.3 TEST MODIFICATIONS.....	2-5
CHAPTER 3.....	3-1
PROCESSING INFORMATION.....	3-1
3.1 TESTING ENVIRONMENT.....	3-1
3.2 SUMMARY OF TEST RESULTS.....	3-1
3.3 TEST EXECUTION.....	3-2
APPENDIX A.....	A-1
MACRO PARAMETERS.....	A-1
APPENDIX B.....	B-1
COMPILATION SYSTEM OPTIONS.....	B-1
LINKER OPTIONS.....	B-2
APPENDIX C.....	C-1
APPENDIX F OF THE Ada STANDARD.....	C-1

CHAPTER 1

INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro92] against the Ada Standard [Ada83] using the current Ada Compiler Validation Capability (ACVC). This Validation Summary Report (VSR) gives an account of the testing of this Ada implementation. For any technical terms used in this report, the reader is referred to [Pro92]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG89].

1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Certification Body may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject implementation has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from the AVF which performed this validation or from:

National Technical Information Service
5285 Port Royal Road
Springfield, Virginia 22161
U.S.A.

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to:

Ada Validation Organization
Computer and Software Engineering Division
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria, Virginia 22311-1772
U.S.A.

1.2 REFERENCES

- [Ada83] Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
- [Pro92] Ada Compiler Validation Procedures, Version 3.1, Ada Joint Program Office, August 1992.
- [UG89] Ada Compiler Validation Capability User's Guide, 21 June 1989.

1.3 ACVC TEST CLASSES

Compliance of Ada implementations is tested by means of the ACVC. The ACVC contains a collection of test programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable. Class B and class L tests are expected to produce errors at compile time and link time, respectively.

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages REPORT and SPRT13, and the procedure CHECK_FILE are used for this purpose. The package REPORT also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The package SPRT13 is used by many tests for Chapter 13 of the Ada Standard. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. If these units are not operating correctly, validation testing is discontinued.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that all violations of the Ada Standard are detected. Some of the class B tests contain legal Ada code which must not be flagged illegal by the compiler. This behavior is also verified.

Class L tests check that an Ada implementation correctly detects violation of the Ada Standard involving multiple, separately compiled units. Errors are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by implementation-specific values--for example, the

largest integer. A list of the values used for this implementation is provided in Appendix A. In addition to these anticipated test modifications, additional changes may be required to remove unforeseen conflicts between the tests and implementation-dependent characteristics. The modifications required for this implementation are described in section 2.3.

For each Ada implementation, a customized test suite is produced by the AVF. This customization consists of making the modifications described in the preceding paragraph, removing withdrawn tests (see section 2.1) and, possibly some inapplicable tests (see Section 3.2 and [UG89]).

In order to pass an ACVC an Ada implementation must process each test of the customized test suite according to the Ada Standard.

1.4 DEFINITION OF TERMS

Ada Compiler	The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof.
Ada Compiler Validation Capability (ACVC)	The means for testing compliance of Ada implementations, Validation consisting of the test suite, the support programs, the ACVC Capability User's Guide and the template for the validation summary (ACVC) report.
Ada Implementation	An Ada compiler with its host computer system and its target computer system.
Ada Joint Program Office (AJPO)	The part of the certification body which provides policy and guidance for the Ada certification Office system.
Ada Validation Facility (AVF)	The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation.
Ada Validation Organization (AVO)	The part of the certification body that provides technical guidance for operations of the Ada certification system.
Compliance of an Ada Implementation	The ability of the implementation to pass an ACVC version.

Computer System	A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units.
Conformity	Fulfillment by a product, process, or service of all requirements specified.
Customer	An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or attainable on the Ada implementation for which validation status is realized.
Host Computer System	A computer system where Ada source programs are transformed into executable form.
Inapplicable Test	A test that contains one or more test objectives found to be irrelevant for the given Ada implementation.
ISO	International Organization for Standardization.
LRM	The Ada standard, or Language Reference Manual, published as ANSI/MIL-STD-1815A -1983 and ISO 8652-1987. Citations from the LRM take the form "<section>.<subsection>:<paragraph>."
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.
Target Computer System	A computer system where the executable form of Ada programs are executed.

Validated Ada Compiler	The compiler of a validated Ada implementation.
Validated Ada Implementation	An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro92].
Validation	The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation.
Withdrawn Test	A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 WITHDRAWN TESTS

Some tests are withdrawn by the AVO from the ACVC because they do not conform to the Ada Standard. The following 104 tests had been withdrawn by the Ada Validation Organization (AVO) at the time of validation testing. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 93-11-22.

B27005A	E28005C	B28006C	C32203A	C34006D	C35507K
C35507L	C35507N	C35507O	C35507P	C35508I	C35508J
C35508M	C35508N	C35702A	C35702B	C37310A	B41308B
C43004A	C45114A	C45346A	C45612A	C45612B	C45612C
C45651A	C46022A	B49008A	B49008B	A54B02A	C55B06A
A74006A	C74308A	B83022B	B83022H	B83025B	B83025D
B83026B	C83026A	C83041A	B85001L	C86001F	C94021A
C97116A	C98003B	BA2011A	CB7001A	CB7001B	CB7004A
CC1223A	BC1226A	CC1226B	BC3009B	BD1B02B	BD1B06A
AD1B08A	BD2A02A	CD2A21E	CD2A23E	CD2A32A	CD2A41A
CD2A41E	CD2A87A	CD2B15C	BD3006A	BD4008A	CD4022A
CD4022D	CD4024B	CD4024C	CD4024D	CD4031A	CD4051D
CD5111A	CD7004C	ED7005D	CD7005E	AD7006A	CD7006E
AD7201A	AD7201E	CD7204B	AD7206A	BD8002A	BD8004C
CD9005A	CD9005B	CDA201E	CE2107I	CE2117A	CE2117B
CE2119B	CE2205B	CE2405A	CE3111C	CE3116A	CE3118A
CE3411B	CE3412B	CE3607B	CE3607C	CE3607D	CE3812A
CE3814A	CE3902B				

2.2 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant for a given Ada implementation. The inapplicability criteria for some tests are explained in documents issued by ISO and the AJPO known as Ada Commentaries and commonly referenced in the format AI-ddddd. For this implementation, the following tests were determined to be inapplicable for the reasons indicated; references to Ada Commentaries are included as appropriate.

C24113I..X (16 TESTS) use a line length in the input file that exceeds 132 characters.

The following 19 tests have floating-point type declarations requiring more digits than `SYSTEM.MAX_DIGITS`:

C24113Y	C35705Y
C35706Y	C35707Y
C35708Y	C35802Y..Z (2 tests)
C45241Y	C45321Y
C45421Y	C45521Y..Z (2 tests)
C45524Y..Z (2 tests)	C45621Y..Z (2 tests)
C45641Y	C46012Y..Z (2 tests)

The following 21 tests check for the predefined type `SHORT_INTEGER`; for this implementation, there is no such type:

C35404B	B36105C	C45231B	C45304B	C45411B
C45412B	C45502B	C45503B	C45504B	C45504E
C45611B	C45613B	C45614B	C45631B	C45632B
B52004E	C55B07B	B55B09D	B86001V	C86006D
CD7101E				

The following 20 tests check for the predefined type `LONG_INTEGER`; for this implementation, there is no such type:

C35404C	C45231C	C45304C	C45411C	C45412C
C45502C	C45503C	C45504C	C45504F	C45611C
C45613C	C45614C	C45631C	C45632C	B52004D
C55B07A	B55B09C	B86001W	C86006C	CD7101F

C35404D, C45231D, B86001X, C86006E, and CD7101G check for a predefined integer type with a name other than `INTEGER`, `LONG_INTEGER`, or `SHORT_INTEGER`; for this implementation, there is no such type.

C35713B, C45423B, B86001T, and C86006H check for the predefined type `SHORT_FLOAT`; for this implementation, there is no such type.

C35713D and B86001Z check for a predefined floating-point type with a name other than `FLOAT`, `LONG_FLOAT`, or `SHORT_FLOAT`; for this implementation, there is no such type.

C45624A..B (2 tests) check that the proper exception is raised if `MACHINE_OVERFLOW` is `FALSE` for floating point types and the results of various floating-point operations lie outside the range of the base type; for this implementation, `MACHINE_OVERFLOW` is `TRUE`.

C4A013B contains a static universal real expression that exceeds the range of this implementation's largest floating-point type; this expression is rejected by the compiler.

B86001Y uses the name of a predefined fixed-point type other than type `DURATION`; for this implementation, there is no such type.

C96005B uses values of type DURATION's base type that are outside the range of type DURATION; for this implementation, the ranges are the same.

CA2009C and CA2009F check whether a generic unit can be instantiated before its body (and any of its subunits) is compiled; this implementation creates a dependence on generic units as allowed by AI-00408 and AI-00506 such that the compilation of the generic unit bodies makes the instantiating units obsolete. (See section 2.3.)

CD1009C checks whether a length clause can specify a non-default size for a floating-point type; this implementation does not support such sizes.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A84O use length clauses to specify non-default sizes for access types; this implementation does not support such sizes.

BD8001A, BD8003A, BD8004A..B (2 tests), and AD8011A use machine code insertions; this implementation provides no package MACHINE_CODE.

AE2101C and EE2201D..E (2 tests) use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types with discriminants without defaults; these instantiations are rejected by this compiler.

AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types with discriminants without defaults; these instantiations are rejected by this compiler.

The 18 tests listed in the following table check that USE_ERROR is raised if the given file operations are not supported for the given combination of mode and access method; this implementation supports these operations.

Test	File Operation	Mode	File Access Method
CE2102E	CREATE	OUT_FILE	SEQUENTIAL_IO
CE2102F	CREATE	INOUT_FILE	DIRECT_IO
CE2102J	CREATE	OUT_FILE	DIRECT_IO
CE2102N	OPEN	IN_FILE	SEQUENTIAL_IO
CE2102O	RESET	IN_FILE	SEQUENTIAL_IO
CE2102P	OPEN	OUT_FILE	SEQUENTIAL_IO
CE2102Q	RESET	OUT_FILE	SEQUENTIAL_IO
CE2102R	OPEN	INOUT_FILE	DIRECT_IO
CE2102S	RESET	INOUT_FILE	DIRECT_IO
CE2102T	OPEN	IN_FILE	DIRECT_IO
CE2102U	RESET	IN_FILE	DIRECT_IO

CE2102V	OPEN	OUT_FILE	DIRECT_IO
CE2102W	RESET	OUT_FILE	DIRECT_IO
CE3102F	RESET	Any_Mode	TEXT_IO
CE3102G	DELETE	-----	TEXT_IO
CE3102I	CREATE	OUT_FILE	TEXT_IO
CE3102J	OPEN	IN_FILE	TEXT_IO
CE3102K	OPEN	OUT_FILE	TEXT_IO

The 3 tests listed in the following table check the given file operations for the given combination of mode and access method; this implementation does not support these operations.

Test	File Operation	Mode	File Access Method
CE2105A	CREATE	IN_FILE	SEQUENTIAL_IO
CE2105B	CREATE	IN_FILE	DIRECT_IO
CE3109A	CREATE	IN_FILE	TEXT_IO

CE2107A..D (4 tests), CE2110B, and CE2111D check operations on sequential files when multiple internal files are associated with the same external file; USE_ERROR is raised when this association is attempted.

CE2107E checks operations on direct and sequential files when files of both kinds are associated with the same external file and both are open for writing; USE_ERROR is raised when this association is attempted.

CE2107H, and CE2107L apply function NAME to temporary sequential, direct, and text files in an attempt to associate multiple internal files with the same external file; USE_ERROR is raised because temporary files have no name.

CE2108B, CE2108D, and CE3112B use the names of temporary sequential, direct, and text files that were created in other tests in order to check that the temporary files are not accessible after the completion of those tests; for this implementation, temporary files have no name.

CE2203A checks that WRITE raises USE_ERROR if the capacity of an external sequential file is exceeded; this implementation cannot restrict file capacity.

CE2403A checks that WRITE raises USE_ERROR if the capacity of an external direct file is exceeded; this implementation cannot restrict file capacity.

CE3111A..B, CE3111D..E, CE3114B, and CE3115A (6 tests) check operations on text files when multiple internal files are associated with the same external file; USE_ERROR is raised when this association is attempted.

CE3304A checks that SET_LINE_LENGTH and SET_PAGE_LENGTH raise USE_ERROR if they specify an inappropriate value for the external file; there are no inappropriate values for this implementation.

CE3413B checks that PAGE raises LAYOUT_ERROR when the value of the page number exceeds COUNT'LAST; for this implementation, the value of COUNT'LAST is greater than 150000, making the checking of this objective impractical.

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 68 tests.

The following tests were split into two or more tests because this implementation did not report the violations of the Ada Standard in the way expected by the original tests.

B22003A	B26001A	B26002A	B26005A	B28003A	B29001A	B33301B
B35101A	B37106A	B37301B	B37302A	B38003A	B38003B	B38009A
B38009B	B55A01A	B61001C	B61001F	B61001H	B61001I	B61001M
B61001R	B61001W	B67001H	B83A07A	B83A07B	B83A07C	B83E01C
B83E01D	B83E01E	B85001D	B85008D	B91001A	B91002A	B91002B
B91002C	B91002D	B91002E	B91002F	B91002G	B91002H	B91002I
B91002J	B91002K	B91002L	B95030A	B95061A	B95061F	B95061G
B95077A	B97103E	B97104G	BA1001A	BA1101B	BC1109A	BC1109C
BC1109D	BC1202A	BC1202F	BC1202G	BE2210A	BE2413A	

C83030C and C86007A were graded passed by Test Modification as directed by the AVO. These tests were modified by inserting "PRAGMA ELABORATE (REPORT);" before the package declarations at lines 13 and 11, respectively. Without the pragma, the packages may be elaborated prior to package Report's body, and thus the packages' calls to function REPORT.IDENT_INT at lines 14 and 13, respectively, will raise PROGRAM_ERROR.

CA2009C and CA2009F were graded inapplicable by Evaluation Modification as directed by the AVO. These tests contain instantiations of a generic unit prior to the compilation of that unit's body; as allowed by AI-00408 and AI-00506, the compilation of the generic unit bodies makes the compilation unit that contains the instantiations obsolete.

BC3204C and BC3205D were graded passed by Processing Modification as directed by the AVO. These tests check that instantiations of generic units with unconstrained types as generic actual parameters are illegal if the generic bodies contain uses of the types that require a constraint. However, the generic bodies are compiled

after the units that contain the instantiations, and this implementation creates a dependence of the instantiating units on the generic units as allowed by AI-00408 and AI-00506 such that the compilation of the generic bodies makes the instantiating units obsolete--no errors are detected. The processing of these tests was modified by re-compiling the obsolete units; all intended errors were then detected by the compiler.

CHAPTER 3

PROCESSING INFORMATION

3.1 TESTING ENVIRONMENT

The Ada implementation tested in this validation effort is described adequately by the information given in the initial pages of this report.

If the point of contact for sales is the same as that for technical use the second option:

For technical information about this Ada implementation, contact:

Mr. Henri T. (Hans) Koppen
Control Data Systems, Inc
4201 Lexington Avenue North
Arden Hills, MN 55126
VOICE: 612-482-4320
FAX: 612-482-4746

For sales information about this Ada implementation, contact:

Mr. Jacques R. Lasserre
Control Data Systems, Inc
5101 Patrick Henry Drive
Santa Clara, CA 95054
VOICE: 408-496-4352
FAX: 408-496-4106

Testing of this Ada implementation was conducted at the customer's site by a validation team from the AVF.

3.2 SUMMARY OF TEST RESULTS

An Ada Implementation passes a given ACVC version if it processes each test of the customized test suite in accordance with the Ada Programming Language Standard, whether the test is applicable or inapplicable; otherwise, the Ada Implementation fails the ACVC [Pro92].

For all processed tests (inapplicable and applicable), a result was obtained that conforms to the Ada Programming Language Standard.

The list of items below gives the number of ACVC tests in various

categories. All tests were processed, except those that were withdrawn because of test errors (item b; see section 2.1), those that require a floating-point precision that exceeds the implementation's maximum precision (item e; see section 2.2), and those that depend on the support of a file system--if none is supported (item d). All tests passed, except those that are listed in sections 2.1 and 2.2 (counted in items b and f, below).

a) Total Number of Applicable Tests	3912	
b) Total Number of Withdrawn Tests	104	
c) Processed Inapplicable Tests	154	
d) Non-Processed I/O Tests	0	
e) Non-Processed Floating-Point Precision Tests	0	
f) Total Number of Inapplicable Tests	154	(c+d+e)
g) Total Number of Tests for ACVC 1.11	4170	(a+b+f)

3.3 TEST EXECUTION

A magnetic tape containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the magnetic tape were loaded directly onto the host/target computer.

After the test files were loaded onto the host/target computer, the full set of tests was processed by the Ada implementation.

The tests were compiled, linked, and executed on the host/target computer system.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B for a complete listing of the processing options for this implementation. It also indicates the default options. The default options invoked for validation testing during this test were:

```
PL (name_of_program_library) DA=NONE EL=W LO=S OL=LOW SC=NONE
UL=TRUE
```

Test output, compiler and linker listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

APPENDIX A

MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG89]. The parameter values are presented in two tables. The first table lists the values that are defined in terms of the maximum input-line length, which is the value for \$MAX_IN_LEN--also listed here. These values are expressed here as Ada string aggregates, where "V" represents the maximum input-line length.

Macro Parameter	Macro Value
\$MAX_IN_LEN	132 -- Value of V
\$BIG_ID1	(1..V-1 => 'A', V => '1')
\$BIG_ID2	(1..V-1 => 'A', V => '2')
\$BIG_ID3	(1..V/2 => 'A') & '3' & (1..V-1-V/2 => 'A')
\$BIG_ID4	(1..V/2 => 'A') & '4' & (1..V-1-V/2 => 'A')
\$BIG_INT_LIT	(1..V-3 => '0') & "298"
\$BIG_REAL_LIT	(1..V-5 => '0') & "690.0"
\$BIG_STRING1	'"' & (1..V/2 => 'A') & '"'
\$BIG_STRING2	'"' & (1..V-1-V/2 => 'A') & '1' & '"'
\$BLANKS	(1..V-20 => ' ')
\$MAX_LEN_INT_BASED_LITERAL	"2:" & (1..V-5 => '0') & "11:"
\$MAX_LEN_REAL_BASED_LITERAL	"16:" & (1..V-7 => '0') & "F.E:"
\$MAX_STRING_LITERAL	'"' & (1..V-2 => 'A') & '"'

The following table contains the values for the remaining macro parameters.

Macro Parameter	Macro Value
ACC_SIZE	: 64
ALIGNMENT	: 1
COUNT_LAST	: 9223372036854775807
DEFAULT_MEM_SIZE	: 134217728
DEFAULT_STOR_UNIT	: 64
DEFAULT_SYS_NAME	: CYBER180
DELTA_DOC	: 2#1.0#E-63
ENTRY_ADDRESS	: NO_SUCH_ENTRY_ADDRESS
ENTRY_ADDRESS1	: NO_SUCH_ENTRY_ADDRESS
ENTRY_ADDRESS2	: NO_SUCH_ENTRY_ADDRESS
FIELD_LAST	: 67
FILE_TERMINATOR	: NO_SUCH_FILE_TERMINATOR
FIXED_NAME	: NO_SUCH_FIXED_TYPE
FLOAT_NAME	: NO_SUCH_FLOAT_TYPE
FORM_STRING	: ""
FORM_STRING2	:
	"CANNOT RESTRICT FILE_CAPACITY"
GREATER_THAN_DURATION	: 100_000_000.0
GREATER_THAN_DURATION_BASE_LAST	: 7_000_000_000.0
GREATER_THAN_FLOAT_BASE_LAST	: 1.80141E+3008
GREATER_THAN_FLOAT_SAFE_LARGE	: 5.221944407067E1232
GREATER_THAN_SHORT_FLOAT_SAFE_LARGE	: NO_SHORT_FLOAT
HIGH_PRIORITY	: 127
ILLEGAL_EXTERNAL_FILE_NAME1	: BADCHAR^@.-!
ILLEGAL_EXTERNAL_FILE_NAME2	:
	MUCH_TOO_LONG_NAME_FOR_A_VE_FILE
INAPPROPRIATE_LINE_LENGTH	: -1
INAPPROPRIATE_PAGE_LENGTH	: -1
INCLUDE_PRAGMA1	:
	PRAGMA INCLUDE ("A28006D1.TST")
INCLUDE_PRAGMA2	:
	PRAGMA INCLUDE ("B28006E1.TST")
INTEGER_FIRST	: -9223372036854775808
INTEGER_LAST	: 9223372036854775807
INTEGER_LAST_PLUS_1	: 9223372036854775808
INTERFACE_LANGUAGE	: FORTRAN
LESS_THAN_DURATION	: -100_000_000.0
LESS_THAN_DURATION_BASE_FIRST	: -7_000_000_000.0
LINE_TERMINATOR	: , 7
LOW_PRIORITY	: 0
MACHINE_CODE_STATEMENT	: NULL;
MACHINE_CODE_TYPE	: NO_SUCH_TYPE
MANTISSA_DOC	: 63
MAX_DIGITS	: 28

MAX_INT	:	9223372036854775807
MAX_INT_PLUS_1	:	9223372036854775808
MIN_INT	:	-9223372036854775808
NAME	:	NO_SUCH_TYPE_AVAILABLE
NAME_LIST	:	CYBER180
NAME_SPECIFICATION1	:	:V07.ADA.ACVC_TEMP.X2120A.;1
NAME_SPECIFICATION2	:	:V07.ADA.ACVC_TEMP.X2120B.;1
NAME_SPECIFICATION3	:	:V07.ADA.ACVC_TEMP.X3119A.;1
NEG_BASED_INT	:	16#FFFF_FFFF_FFFF_FFF8#
NEW_MEM_SIZE	:	134_217_728
NEW_STOR_UNIT	:	64
NEW_SYS_NAME	:	CYBER180
PAGE_TERMINATOR	:	' '
RECORD_DEFINITION	:	NEW_INTEGER;
RECORD_NAME	:	NO_SUCH_MACHINE_CODE_TYPE
TASK_SIZE	:	64
TASK_STORAGE_SIZE	:	64
TICK	:	0.001
VARIABLE_ADDRESS	:	VARIABLE'ADDRESS
VARIABLE_ADDRESS1	:	VARIABLE1'ADDRESS
VARIABLE_ADDRESS2	:	VARIABLE2'ADDRESS
YOUR_PRAGMA	:	COMMON, EXPORT

APPENDIX B

COMPILATION SYSTEM OPTIONS

The compiler options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report.

This chapter explains the NOS/VE Ada compiling, linking, and execution processes. A sample SCL procedure is provided at the end of the chapter to run these processes automatically.

Compiler Command

NOS/VE Ada Implementation Feature

The ADA compiler command compiles your source program and, upon successful compilation, stores one or more library units into the specified program library. The ADA command uses the NOS/VE parameter conventions described in appendix D.

ADA

Purpose Invokes the compiler and specifies the current sublibrary, the files to be used, and the compiler options to be used.

Format **ADA**
 INPUT=file
 PROGRAM _LIBRARY=file
 LIST=file
 DEBUG _AIDS=keyword
 ERROR=file
 ERROR _LEVEL=keyword
 LIST _OPTIONS=keyword
 OPTIMIZATION _LEVEL=keyword
 SUPPRESS _CHECKS=list of keyword
 UPDATE _LIBRARY=boolean
 STATUS=status variable

Parameters **INPUT (I)**

File that contains the source text to be read. The source input ends when an end-of-partition or an end-of-information is encountered on the source input file. The default value is \$INPUT.

PROGRAM _LIBRARY (PL)

Name of the current sublibrary. The default is \$USER.ADA_PROGRAM_LIBRARY. See Getting Started in chapter 2, if you have not set up an Ada program library.

LIST (L)

File where the compiler writes the source listing, diagnostics, statistics, and any additional list information specified by the LIST_OPTIONS parameter. The default value is \$LIST, which, by default, is connected to \$NULL.

NOTE

You can redirect your \$LIST with the CREATE_FILE_CONNECTIONS (CREFC) command. For example:

```
/create_file_connection standard_file=$list file=output_file
```

This connection stays active until you DELETE_FILE_CONNECTION or logout. See the NOS/VE System Usage manual for more information.

DEBUG_AIDS (DA)

Debug options to be used.

Keyword	Description
ALL	All of the available Debug options are selected.
DT	Generates a line number table as part of the object code. This line number table is used by Debug during traceback.
NONE	No Debug tables are produced. If this option is selected, the program cannot be executed under Debug control.

If the parameter is omitted, NONE is assumed.

ERROR (E)

File to receive the error listing. The default value is \$ERRORS.

ERROR_LEVEL (EL)

Minimum severity level of the diagnostics to be listed. The levels, in increasing order of severity, are:

Keyword	Description
I	INFORMATIONAL. The syntax of the construct is correct but the usage is questionable.
W	WARNING. An error that does not change the meaning of the program or hinder the generation of object code. Also, a construct for which the object code raises a CONSTRAINT_ERROR at run time.
F	FATAL. An illegal construct in the source program has been detected. The compilation continues, but no object code is generated.
C	CATASTROPHIC. An error that causes the compiler to be terminated immediately. No object code is generated.

If the parameter is omitted, W is assumed so all diagnostics are listed.

NOTE

You can redirect your \$LIST with the CREATE_FILE_CONNECTIONS (CREFC) command. For example:

```
/create_file_connection standard_file=$list file=output_file
```

This connection stays active until you DELETE_FILE_CONNECTION or logout. See the NOS/VE System Usage manual for more information.

DEBUG_AIDS (DA)

Debug options to be used.

Keyword	Description
ALL	All of the available Debug options are selected.
DT	Generates a line number table as part of the object code. This line number table is used by Debug during traceback.
NONE	No Debug tables are produced. If this option is selected, the program cannot be executed under Debug control.

If the parameter is omitted, NONE is assumed.

ERROR (E)

File to receive the error listing. The default value is \$ERRORS.

ERROR_LEVEL (EL)

Minimum severity level of the diagnostics to be listed. The levels, in increasing order of severity, are:

Keyword	Description
I	INFORMATIONAL. The syntax of the construct is correct but the usage is questionable.
W	WARNING. An error that does not change the meaning of the program or hinder the generation of object code. Also, a construct for which the object code raises a CONSTRAINT_ERROR at run time.
F	FATAL. An illegal construct in the source program has been detected. The compilation continues, but no object code is generated.
C	CATASTROPHIC. An error that causes the compiler to be terminated immediately. No object code is generated.

If the parameter is omitted, W is assumed so all diagnostics are listed.

LIST_OPTIONS (LO)

Information written to the listing file (LIST parameter). Multiple options can be specified; for example, LO=(O,S).

Keyword	Description
O	Object and source code listing.
R	Symbolic cross-reference listing of all program entities.
S	Source input listing.
ALL	All of the available list options.
NONE	No list options are selected.

If the parameter is omitted, S is assumed.

OPTIMIZATION_LEVEL (OL)

Level of object code optimization.

Keyword	Description
LOW	Lowest level of production quality code. No optimization is performed.
DEBUG	Generates code to support step mode debugging.

If the parameter is omitted, LOW is assumed. See chapter 8 for information about debugging.

SUPPRESS_CHECKS (SC)

Specifies runtime checks to be suppressed in the same manner as if an explicit pragma **SUPPRESS** had been inserted in the source code for the compilation unit. The pragma **SUPPRESS** and detailed descriptions of the runtime checks are given in the Ada reference manual.

Keyword	Description
ALL	Suppresses all runtime checks.
ACCESS (A)	Suppresses the check for null values in referenced pointers.
DISCRIMINANT (D)	Suppresses the check that fields of a record exist for the value of a discriminant.
ELABORATION (E)	Suppresses the check that the elaboration before access rule has been obeyed.
INDEX (I)	Suppresses the check that an array index is within bounds.
LENGTH (L)	Suppresses the check of array lengths for array operations.
RANGE (R)	Suppresses the check that a value of a scalar remains within the bounds defined for its type or subtype.
NONE	No runtime checks are suppressed.

If the parameter is omitted, **NONE** is assumed.

UPDATE_LIBRARY (UL)

Specifies whether or not the compiler is to update the Program Library with the result of the compilation. In either case, the compiler performs a syntax check.

TRUE	A syntactic and semantic check is performed and the result of the compilation is saved in the Program Library.
FALSE	A syntactic and semantic check is performed but the Program Library is not updated.

If the parameter is omitted, **TRUE** is assumed.

STATUS

Specifies the name of the SCL status variable to be set by the compiler at completion time.

Examples The following compile command uses all of the default values:

```
/ada i=my_source
```

The compiled source is written to the default program library \$USER.ADA_PROGRAM_LIBRARY. The listing is output to file \$LIST. Any errors are sent to file \$ERRORS. No debug tables are produced.

The following compile command specifies program library YOURPL:

```
/ada i=your_file pl=yourpl l=list da=all e=error lo=(r,s)
```

The following compile command uses the default program library:

```
/ada i=your_file l=list da=all e=error lo=(r,s)
```

Figure 3-1 presents the DISPLAY_COMMAND_INFORMATION output for ADA.

```
/disc1 ada
input, i           : file = $INPUT
program_library, pl : file = $USER.ADA_PROGRAM_LIBRARY
list, l           : file = $LIST
debug_aids, da     : key none, dt, all, keyend = none
error, e          : file = $ERRORS
error_level, el    : key (informational, i), (warning, w),
                   : (fatal, f), (catastrophic, c), keyend = w
list_options, lo   : list of key none, o, r, s, all, keyend = s
optimization_level, ol : key debug, low, high, keyend = low
suppress_checks, sc : (discriminant, d), (elaboration, e),
                   : (index, i), (length, l), (range, r),
                   : none, keyend = none
update_library, ul  : boolean = true
status             : (VAR, BY_NAME) status = $optional
```

Figure 3-1. Sample DISPLAY_COMMAND_INFORMATION Output

For Better Performance

When multiple compilation units are submitted, performance is better if they are included in a single file rather than on multiple files. However, if the number of compilation units grows over a certain limit (for example, 50 small compilation units of about 50 lines each) or if the first compilation units are large, a degradation of the throughput actually occurs.

End of NOS/VE Ada Implementation Feature

LINKER OPTIONS

The linker options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to linker documentation and not to this report.

Linker Command

NOS/VE Ada Implementation Feature

The Ada linker command links your compilation units or, if the RECOMPILATIONS parameter is specified, checks the recompilation dependencies of one or more compilation units. The LINK_ADA command uses the NOS/VE syntax conventions described in appendix D.

LINK_ADA

Purpose Links Ada code after it has been compiled and before it can be executed.

Format LINK_ADA or LINA
 - MAIN_PROGRAM=name
 - PROGRAM_LIBRARY=file
 BINARY=file
 LIST=file
 RECOMPILATIONS=string
 STATUS=status variable

Parameters MAIN_PROGRAM (MP)

Compilation unit to be linked, that is, the name of the procedure to be linked. It must be a parameterless procedure. The procedure must have been compiled so that it is a library unit in the sublibrary specified by the PROGRAM_LIBRARY parameter. The compilation unit names can be listed using the SHOW command in a PLU session. This parameter is required.

PROGRAM_LIBRARY (PL)

File containing the sublibrary to be referenced by the linker. The default value is \$USER.ADA_PROGRAM_LIBRARY.

BINARY (B)

File on which the executable code extracted from the user's program library is written, thus creating an object file acceptable to the NOS/VE loader. If \$NULL is specified, the Ada linker performs all the compilation order validation checks, but does not create an object file. The default value is \$LOCAL.LGO.

LIST (L)

File where the linker writes the library units elaboration order list. The default value is \$LIST, which, by default, is connected to \$NULL.

NOTE

You can redirect your \$LIST with the CREATE_FILE_CONNECTIONS (CREFC) command. For example:

```
/create_file_connection standard_file=$list file=output_file
```

This connection stays active until you DELETE_FILE_CONNECTION or logout. See the NOS/VE System Usage manual for more information.

RECOMPILATIONS (R)

Name or names of any modules that need to be recompiled. This parameter must be omitted to produce a binary file.

STATUS

Name of the SCL status variable in which the linker stores its termination condition at completion time.

Remarks The main program name for the Ada linker is a procedure name used in the source text.

NOTE

The main program for the LINK_ADA command must be a parameterless procedure.

The default binary file name, \$LOCAL.LGO, is also the default file name for the EXECUTE_TASK command.

Examples The following link command produces a list of dependencies. A binary file is not produced:

```
/link_ada main_program=your_procedure recompilations ..
../list=dependencies_list
```

The following link command produces a binary file. A list of dependencies is not produced:

```
/link_ada mp=your_procedure binary=binary_file
```

Figure 3-2 presents the DISPLAY_COMMAND_INFORMATION output for LINK_ADA.

```
/disc1 link_ada
main_program, mp      : name = $required
program_library, pl   : file = $USER.ADA_PROGRAM_LIBRARY
binary, b             : file = $LOCAL.lgo
list, l               : file = $LIST
recompilations, r     : string = $optional
status                : var of status = $optional
```

Figure 3-2. Sample DISPLAY_COMMAND_INFORMATION Output

***** End of NOS/VE Ada Implementation Feature *****

Execution

NOS/VE Ada Implementation Feature

Once you have compiled and linked an Ada program, the linked binary can be loaded and executed in the following ways:

- Using the EXECUTE_TASK command, as described in the NOS/VE Commands and Functions manual
- Using a direct file reference to the binary file produced by the Ada linker
- Using a call to the system interface procedure PMP\$EXECUTE from within another program, as described in the CYBIL System Interface manual

For Better Performance

Use of the INLINE pragma, where applicable, results in faster object code by avoiding the call/return instructions and the dynamic initialization of the stack frame.

Using the EXECUTE_TASK Command

The EXECUTE_TASK command offers the most flexibility in loading and executing your program. After compiling your program file and linking your compilation units, you enter the following command at the system prompt:

```
/execute_task file=your_binary_file_name
```

For example:

```
/exet f=math1bgo
```

Or, if you had omitted the BINARY parameter from the LINK_ADA command:

```
/exet $local.lgo
```

Or the equivalent (if your working catalog is \$LOCAL):

```
/lgo
```

APPENDIX C

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

package STANDARD is

type INTEGER is range -9223372036854775808 .. 9223372036854775807;

type FLOAT is digits 13

range -16#7.FFFF_FFFF_FFF8#E1023 .. 16#7.FFFF_FFFF_FFF8#E1023;

type LONG_FLOAT is digits 28

range -16#7.FFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFF8#E1023 ..
16#7.FFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFF8#E1023;

type DURATION is delta 0.001 range -6_279_897_600.0 ..
6_279_897_600.0;

end STANDARD;

Implementation-Dependent Characteristics F

F.1 NOS/VE Ada Pragmas	F-2
F.2 NOS/VE Ada Attributes	F-3
F.3 Specification of the Package SYSTEM	F-3
F.4 Restrictions on Representation Clauses	F-3
F.4.1 Length Clauses	F-4
F.4.2 Enumeration Representation Clauses	F-4
F.4.3 Record Representation Clauses	F-4
F.5 Implementation-Dependent Names	F-5
F.6 Address Clauses and Interrupts	F-5
F.7 Unchecked Type Conversions	F-5
F.8 Input-Output Packages	F-5
F.8.1 External Files and File Objects	F-6
F.8.2 Exceptions for Input-Output Errors	F-6
F.8.3 Low Level Input-Output	F-7
F.9 Other Implementation-Dependent Characteristics	F-7
F.9.1 Implementation Features	F-7
F.9.1.1 Predefined Types	F-7
F.9.1.2 Basic Types	F-8
F.9.1.3 Compiler	F-8
F.9.1.4 Definition of a Main Program	F-9
F.9.1.5 TIME Type	F-9
F.9.1.6 Machine Code Insertions	F-9
F.9.2 Entity Types	F-9
F.9.2.1 Array Types	F-9
F.9.2.2 Record Types	F-10
F.9.2.2.1 Simple Record Types (Without Discriminants)	F-10
F.9.2.2.2 Record Types With Discriminants	F-11
F.9.2.3 Access Types	F-11
F.9.3 Tasking	F-11
F.9.4 Interfaces to Other Languages	F-11
F.9.5 Command Interfaces	F-11
F.9.5.1 Program Library Utility Commands	F-12
F.9.5.2 Compiler Command	F-12
F.9.5.3 Linker Command	F-12
F.9.5.4 Execution	F-12
F.9.6 Values of Data Attributes	F-12
F.9.6.1 Values of Integer Attributes	F-13
F.9.6.2 Values of Floating Point Attributes	F-13
F.9.6.3 Values of Long Floating Point Attributes	F-14
F.9.6.4 Values of Duration Attributes	F-14

Implementation-Dependent Characteristics F

This appendix summarizes the implementation-dependent characteristics of NOS/VE Ada by listing the following:

- NOS/VE Ada pragmas
- NOS/VE Ada attributes
- Specification of the package SYSTEM
- Restrictions on representation clauses
- Implementation-dependent names
- Address clauses and interrupts
- Unchecked type conversions
- Input-output packages
- Other implementation-dependent characteristics

Shading is not used in this appendix.

F.1 NOS/VE Ada Pragmas

NOS/VE Ada supports the following pragmas as described in the ANSI/MIL-STD-1815A-1983, Reference Manual for the Ada Programming Language, except as shown below:

- **INLINE**

This pragma causes inline expansion of a subprogram except as described in annex B of this manual (see 6.3.2, 10.6).

- **INTERFACE**

This pragma is supported for CYBIL, FORTRAN, and the NOS/VE Math Library, as discussed in 13.9.1, 13.9.2, and 13.9.3, respectively.

- **PACK**

Arrays of components of discrete type are packed into the nearest 2^n bits.

- **SHARED**

This pragma is not supported for the following types of variables:

- Variables of type **LONG_FLOAT**
- Variables of a subtype of type **LONG_FLOAT**
- Variables of a type derived from type **LONG_FLOAT**
- Variables of a subtype derived from type **LONG_FLOAT**

- **SUPPRESS**

This pragma is supported, but it is not possible to restrict the check suppression to a specific object or type.

NOS/VE Ada does not support the following pragmas:

- **CONTROLLED**
- **MEMORY_SIZE**
- **OPTIMIZE**
- **STORAGE_UNIT**
- **SYSTEM_NAME**

NOS/VE Ada supports the following implementation-defined pragmas:

- **COMMON**

This pragma accepts the name of a FORTRAN labeled common as its single argument. This pragma is allowed only in the specification of a library package. This pragma specifies that the library package specification can be accessed as a labeled common by a FORTRAN subroutine. To ensure proper results, the items declared in the Ada library package specification must be of a type corresponding to the type of the matching items in the FORTRAN common specification. The argument name must be a legal NOS/VE and FORTRAN name.

- **EXPORT**

This pragma accepts a language name and a subprogram name as arguments. This pragma is allowed only in the body of a library procedure. This pragma specifies the other language (FORTRAN) and informs the Ada compiler that it must provide an entry point in the procedure by the specified subprogram name. (FORTRAN is the only supported language.) The subprogram name must be a NOS/VE and FORTRAN legal name. Parameter passing is not supported.

F.2 NOS/VE Ada Attributes

NOS/VE Ada supports all the attributes described by the ANSI/MIL-STD-1815A-1983, Reference Manual for the Ada Programming Language. It does not provide any implementation-defined attributes. The NOS/VE implementation of the P'ADDRESS attribute returns the prefix P, the 48-bit process virtual address (PVA) right-justified within a 64-bit variable of the predefined type INTEGER.

F.3 Specification of the Package SYSTEM

```
package SYSTEM is
  type ADDRESS is access INTEGER;
  type NAME is (CYBER180);

  SYSTEM_NAME : constant NAME := CYBER180;

  STORAGE_UNIT : constant := 64; -- 64-bit machine
  MEMORY_SIZE : constant := 128*1048576; -- 128 megabytes

  MIN_INT : constant := -9_223_372_036_854_775_808; -- (-2**63)
  MAX_INT : constant := 9_223_372_036_854_775_807; -- (2**63)-1
  MAX_DIGITS : constant := 28;
  MAX_MANTISSA : constant := 63;
  FINE_DELTA : constant := 2#1.0#E-63; -- 2**(-63)
  TICK : constant := 0.001;

  subtype PRIORITY is INTEGER range 0 .. 127;

end SYSTEM;
```

F.4 Restrictions on Representation Clauses

NOS/VE Ada implements representation clauses as described by the ANSI standard for Ada. It does not allow representation clauses for a derived type.

NOS/VE Ada supports the type representation clauses with some restrictions:

- Length clauses
- Enumeration representation clauses
- Record representation clauses
- Address clauses

NOS/VE Ada does not support interrupts.

F.4.1 Length Clauses

NOS/VE Ada supports the attributes in the length clauses as follows:

- **T'SIZE**

NOS/VE Ada accepts the SIZE attribute for a type T under the following conditions:

- If T is a discrete type, then the specified size must be less than or equal to 64 and when the number of bits needed to hold any value of the type is calculated, the range is extended to include 0 if necessary, i.e., the range 3..4 cannot be represented in 1 bit, but needs 3 bits.
- If T is a fixed point type, then the specified size must be less than or equal to 64 bits. As for discrete types, the number of bits needed for a fixed point type is calculated using the range of the fixed point type possibly extended to include 0.0.
- If T is a floating point type, an access type or a task type, the specified size must be equal to the number of bits used to represent values of the type (floating points: 64 or 128 bits, access types: 64 bits and task type: 64 bits).
- If T is a record type, the specified size must be greater than or equal to the minimal number of bits used to represent values of the type per default.
- If T is an array type, the specified size must be equal to the minimal number of bits used to represent values of the type per default.

- **T'STORAGE_SIZE (collection size)**

Supported

- **T'STORAGE_SIZE (task activation size)**

Supported

F.4.2 Enumeration Representation Clauses

In NOS/VE Ada enumeration representation clauses, the internal codes must be in the range of the predefined type INTEGER.

F.4.3 Record Representation Clauses

NOS/VE Ada implements record representation clauses as described by the ANSI language definition.

NOS/VE Ada supports only alignment on storage unit boundary, i.e., at mod 1, in record representation clauses.

The component clause of a record representation clause gives the storage place of a component of a record, by providing the following pieces of data:

- The name gives the name of the record component.
- The simple expression following the reserved word AT gives the address in storage units, relative to the beginning of the record, of the storage unit where the component starts.

- The range in the component clause gives the bit positions, relative to that starting storage unit, occupied by the record component.

NOS/VE Ada supports the range for only those record components of discrete types (integer or enumeration) or arrays of discrete elements. The range must be less or equal to 64 bits and all values of the component type must be representable in the specified number of bits. The component may start at any bit boundary. A range can overlap 2 adjacent storage units.

If the record type contains components which are not covered by a component clause, they are allocated consecutively after the last component with a component clause. Allocation of a record component without a component clause is always aligned on a storage unit boundary. Holes created because of component clauses are not otherwise utilized by the compiler.

F.5 Implementation-Dependent Names

NOS/VE Ada does not support implementation-dependent names to be used in record representation clauses.

F.6 Address Clauses and Interrupts

NOS/VE Ada accepts address clauses for objects but 'address literals' cannot be specified. Address clauses will therefore look like:

for X use Y'ADDRESS;

NOS/VE Ada does not accept address clauses for subprograms, tasks, packages, and entries.

NOS/VE Ada does not support interrupts.

F.7 Unchecked Type Conversions

NOS/VE Ada allows unchecked conversions when objects of the source and target types have the same size.

F.8 Input-Output Packages

The discussion of NOS/VE Ada implementation of input-output packages includes the following:

- External files and file objects
- Exceptions for input-output errors
- Low level input-output

F.8.1 External Files and File Objects

NOS/VE Ada can process files created by another language processor as long as the data types and file structures are compatible.

NOS/VE Ada supports the following kinds of external files:

- Sequential access files (see 14.1)
- Direct access files (see 14.1)
- Text input-output files (see 14.3)

F.8.2 Exceptions for Input-Output Errors

The ANSI/MIL-STD-1815A-1983 Reference Manual for the Ada Programming Language describes conditions under which input-output exceptions are raised. In addition to these, NOS/VE Ada raises the following exceptions:

- The exception `DATA_ERROR` is raised when:
 - An attempt is made to read from a direct file a record that has not been defined.
 - A check reveals that the sizes of the records read from a file do not match the sizes of the Ada variables. NOS/VE Ada performs this check except in those few instances where it is too complicated to do so (see 14.2.2).
- The exception `USE_ERROR` is raised when:
 - The function `NAME` references a temporary file (see 14.2.1).
 - An attempt is made to delete an external direct file with multiple accesses while more than one instance of open is still active. The file remains open and the position is unchanged (see 14.2.1).
 - An attempt is made to create a sequential, text, or direct file of mode `IN_FILE` (see 14.2.1).
 - An attempt is made to create an existing file (see 14.2.1).
 - An attempt is made to process a text file with a line that is longer than 511 characters.
 - An attempt is made to set the page length for a text file that does not have the file contents of `LIST` (see 14.3.3).
 - An attempt is made to issue a new page for a text file that does not have the file contents of `LIST`.
 - An attempt is made to open or create a file with the `FORM` parameter specifying anything other than an empty string for sequential access or direct access files.
 - An attempt is made to set a line for a text file that does not have the file contents of `LIST` and the value specified by `TO` is less than the current line number.

- An attempt is made to open or create a text file with the FORM parameter specifying any other value than LIST, LEGIBLE, or UNKNOWN.
- An attempt is made to open or create a text file with attribute FILE_CONTENTS not matching the file format specified by the FORM parameter.

F.8.3 Low Level Input-Output

NOS/VE Ada does not support the package LOW_LEVEL_IO.

F.9 Other Implementation-Dependent Characteristics

The other implementation-dependent characteristics of NOS/VE Ada are discussed as follows:

- Implementation features
- Entity types
- Tasking
- Interface to other languages
- Command interfaces
- Values of data attributes

F.9.1 Implementation Features

The NOS/VE Ada implementation features are listed as follows:

- Predefined types
- Basic types
- Compiler
- Definition of a main program
- TIME type
- Machine code insertions

F.9.1.1 Predefined Types

NOS/VE Ada implements all the predefined types described by the ANSI/MIL-STD-1815A-1983, Reference Manual for the Ada Programming Language, except:

- LONG_INTEGER
- SHORT_FLOAT
- SHORT_INTEGER

F.9.1.2 Basic Types

The sizes of the basic types are as follows:

Type	Size (bytes)
ENUMERATION	8
FLOAT	8
INTEGER	8
LONG_FLOAT	16
TASK	8

In NOS/VE Ada, the enumeration type includes predefined type boolean and character as well as user defined enumeration types.

F.9.1.3 Compiler

NOS/VE Ada provides an ANSI standard Ada compiler.

The NOS/VE Ada compiler has the following characteristics:

- Source code lines up to 132 characters long
- Up to 100 static levels of nesting of blocks and/or subprograms
- External files up to one segment, $2^{31}-1$ bytes, in length
- A generic body can be compiled in a separate file from its specification if it is compiled before it is instantiated. If the specification, body and instantiation are in the same file, the instantiation of the generic can be either before or after the compilation of the body.
- A generic non-library package body or a generic non-library subprogram body cannot be compiled as a subunit in a separate file from its specification.

For Better Performance

The compiler throughput improves when multiple compilation units are submitted. However, if the number of compilation units grows over a certain limit, for example 50 small compilation units of about 50 lines each, or if the first compilation units are large, the throughput actually degrades.

Using the pragma `INLINE`, where applicable, results in faster object code by avoiding the call/return instructions.

F.9.1.4 Definition of a Main Program

NOS/VE Ada requires that the main program be a procedure without parameters. The name of a compilation unit used as a main program must follow NOS/VE naming standards. The name can be up to 31 characters in length and must be a valid NOS/VE name and a valid Ada identifier. Any naming error is detected at link time only. For more information, see the Ada for NOS/VE Usage manual.

F.9.1.5 TIME Type

NOS/VE Ada defines the type `TIME` as an integer representing the Julian date in milliseconds.

F.9.1.6 Machine Code Insertions

NOS/VE Ada does not support machine code insertions.

F.9.2 Entity Types

This discussion contains information on:

- Array types
- Record types
- Access types

F.9.2.1 Array Types

Arrays are stored row wise, that is, the last index changes the fastest.

An array has a type descriptor that NOS/VE Ada uses when the array is one of the following:

- A component of a record with discriminants
- Passed as a parameter
- Created by an allocator

For each index, NOS/VE Ada builds the following information triplet:

Lower Bound
Upper Bound
Element Size

For multi-dimension arrays, NOS/VE Ada allocates the triplets consecutively.

Element size is expressed in number of storage units (64-bit words). If the array is packed, the element size is expressed in number of bits and represented by a negative value.

NOS/VE Ada strings are packed arrays of characters. Each component of the array is an 8-bit (1-byte) character. Packed arrays of booleans use 1 bit per component and are left-justified. Arrays of integers or enumeration variables can also be packed. Each component uses n bits. Thus, the integer or enumeration subtype is in the range $-2^{**n} .. (2^{**n})-1$.

Note that all objects start on a storage unit (64-bit word) boundary.

At run time when NOS/VE Ada elaborates an array definition, the amount of available space remaining either on the stack or in the heap limits the maximum size of the array (see 3.6).

F.9.2.2 Record Types

At run time when NOS/VE Ada elaborates a record definition, the amount of available space remaining either on the stack or in the heap limits the maximum size of the record (see 3.7).

NOS/VE Ada raises the exception `STORAGE_ERROR` at run time when the size of an elaborated object exceeds the amount of available space.

The rest of this discussion on how records are stored includes:

- Simple record types (without discriminants)
- Record types with discriminants

F.9.2.2.1 Simple Record Types (Without Discriminants)

In the absence of representation clauses, each record component is word aligned. NOS/VE Ada stores the record components in the order they are declared.

A fixed size array (lower and upper bounds are constants) is stored within the record. Otherwise, the array is stored elsewhere in the heap, and is replaced by a pointer to the array value (first element of the array) in the record.

F.9.2.2.2 Record Types With Discriminants

The discriminants are stored first, followed by all the other components as described for simple records.

If a record component is an array with index values that depend on the value of the discriminant(s), the array and its descriptor are both allocated on the heap. They are replaced by a pair of pointers in the record. One points to the array value and the other points to the array descriptor.

F.9.2.3 Access Types

Objects of access type are 6-byte pointers, left-justified within a word, to the accessed data contained in some allocated area in the heap. If the accessed data is of type array or packed array, the allocated area also contains the address of the array descriptor in front of the data.

F.9.3 Tasking

NOS/VE Ada supports tasking by running all Ada tasks as NOS/VE concurrent procedures activated and controlled by the tasking kernel which is an integral part of the NOS/VE compiler run time system. Contact the site administrator to change the site's TASK_LIMIT to run more concurrent tasks than the site currently allows. See the Ada for NOS/VE Usage manual for more information on NOS/VE Ada tasking.

F.9.4 Interfaces to Other Languages

NOS/VE Ada supports calls to CYBIL and FORTRAN subprograms and to NOS/VE Math Library subroutines with the following restrictions:

- CYBIL interface
(See 13.9.1 and chapter 6 of the Ada for NOS/VE Usage manual).
- FORTRAN interface
(See 13.9.2 and chapter 6 of the Ada for NOS/VE Usage manual).
- Math Library interface
(See 13.9.3 and chapter 6 of the Ada for NOS/VE Usage manual).

F.9.5 Command Interfaces

The discussion of the command interfaces implemented by NOS/VE Ada includes:

- Program Library Utility commands
- Compiler command
- Linker command
- Execution commands

NOS/VE Ada commands use the syntax and language elements for parameters described in the NOS/VE System Usage manual.

F.9.5.1 Program Library Utility Commands

NOS/VE Ada provides a hierarchically structured (tree structured) program library to fulfill the ANSI Ada language definition requirements. A node (sublibrary) in the tree can contain up to 4096 compilation units. The Ada for NOS/VE Usage manual contains a detailed discussion of the NOS/VE Ada implementation of the program library.

F.9.5.2 Compiler Command

The NOS/VE Ada compiler can compile an ANSI standard Ada program on NOS/VE. See the Ada for NOS/VE Usage manual for information about the NOS/VE Ada compiler command.

F.9.5.3 Linker Command

The NOS/VE Ada linker checks the order of compilation of the compilation units of a program before the program can be executed.

See the Ada for NOS/VE Usage manual for more information about the linker command.

F.9.5.4 Execution

NOS/VE Ada provides several ways to load and execute an Ada program. They are described in the following manuals:

- Ada for NOS/VE Usage
- CYBIL for NOS/VE System Interface Usage
- NOS/VE Object Code Management Usage

F.9.6 Values of Data Attributes

The package STANDARD contains the declaration of the following predefined types and their attributes:

- Integer (INTEGER)
- Floating point (FLOAT)
- Long floating point (LONG_FLOAT)
- Duration (DURATION)

F.9.6.1 Values of Integer Attributes

Attribute	Value
FIRST	-9_223_372_036_854_775_808
LAST	9_223_372_036_854_775_807
SIZE	64
WIDTH	20

F.9.6.2 Values of Floating Point Attributes

Attribute	Value
DIGITS	13
EMAX	180
EPSILON	5.6_843_419_961#E-14
FIRST	-16#7.FFFF_FFFF_FFF8#E1023
LARGE	1.532495540866E54
LAST	16#7.FFFF_FFFF_FFF8#E1023
MACHINE_EMAX	4095
MACHINE_EMIN	-4096
MACHINE_MANTISSA	48
MACHINE_OVERFLOWS	TRUE
MACHINE_RADIX	2
MACHINE_ROUNDS	FALSE
MANTISSA	45
SAFE_EMAX	4095
SAFE_LARGE	5.221944407066E1232
SAFE_SMALL	9.574977460952E-1234
SIZE	64
SMALL	3.262652233999E-55

F.9.6.3 Values of Long Floating Point Attributes

Attribute	Value
DIGITS	28
EMAX	380
FIRST	-16#7.FFFF_FFFF_FFFF_FFFF_FFFF_FFF8#E1023
LARGE	2.462625387274654950767440006E114
LAST	16#7.FFFF_FFFF_FFFF_FFFF_FFFF_FFF8#E1023
MACHINE_EMAX	4095
MACHINE_EMIN	-4096
MACHINE_MANTISSA	96
MACHINE_OVERFLOWS	TRUE
MACHINE_RADIX	2
MACHINE_ROUNDS	FALSE
MANTISSA	95
SAFE_EMAX	4095
SAFE_LARGE	5.221944407065762533458763552E1232
SAFE_SMALL	9.574977460952185357946731011E-1234
SIZE	128
SMALL	2.030353469852519378619219645E-115

F.9.6.4 Values of Duration Attributes

Attribute	Value
DELTA	1.000000000000E-03
LARGE	8.589934591999E09
MACHINE_OVERFLOWS	TRUE
MACHINE_ROUNDS	FALSE
SIZE	64
SMALL	9.765625000000E-04